

Open Research Online

The Open University's repository of research publications and other research outputs

A framework for feeding Linked Data to Complex Event Processing engines

Conference or Workshop Item

How to cite:

Liu, Dong; Pedrinaci, Carlos and Domingue, John (2010). A framework for feeding Linked Data to Complex Event Processing engines. In: The 1st International Workshop on Consuming Linked Data (COLD 2010) at The 9th International Semantic Web Conference (ISWC 2010), 8 Nov 2010, Shanghai, China.

For guidance on citations see [FAQs](#).

© 2010 The Authors

Version: Accepted Manuscript

Link(s) to article on publisher's website:

http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-665/LiuEtAl_COLD2010.pdf

Copyright and Moral Rights for the articles on this site are retained by the individual authors and/or other copyright owners. For more information on Open Research Online's data [policy](#) on reuse of materials please consult the policies page.

oro.open.ac.uk

A Framework for Feeding Linked Data to Complex Event Processing Engines

Dong Liu, Carlos Pedrinaci, and John Domingue

Knowledge Media Institute, The Open University
Walton Hall, Milton Keynes, MK7 6AA, UK
`{d.liu,c.pedrinaci,j.b.domingue}@open.ac.uk`

Abstract. A huge volume of Linked Data has been published on the Web, yet is not processable by Complex Event Processing (CEP) or Event Stream Processing (ESP) engines. This paper presents a framework to bridge this gap, under which Linked Data are first translated into events conforming to a lightweight ontology, and then fed to CEP engines. The event processing results will also be published back onto the Web of Data. In this way, CEP engines are connected to the Web of Data, and the ontological reasoning is integrated with event processing. Finally, the implementation method and a case study of the framework are presented.

Keywords: Linked Data, Complex Event Processing, ontology mapping, rule-based reasoning.

1 Introduction

With the development of the Semantic Web, a huge volume of Linked Data has been published on the Web. On the other hand, with the rise of Complex Event Processing (CEP) and Event Stream Processing (ESP) [9], steps have been made towards their integration with semantic technologies, i.e. Semantic CEP [5, 15]. However, Linked Data is not processable by CEP or ESP engines for several reasons: i) the existing engines, such as Drools Fusion¹ and Esper² are object-oriented and lack the capability of accessing the Web of Linked Data; ii) although there are a set of tools for generating Java objects from RDF statements³, it is still difficult for CEP engines to manipulate Linked Data, because of the heterogeneity of schemas or ontologies defined by independent data providers; iii) semantic repositories cannot perform temporal reasoning over RDF triples. Extensions to RDF and SPARQL have been made to address these issues [14, 3], but they are not realizable because of the requirement of modifying the semantic repositories or query execution engines.

¹ <http://www.jboss.org/drools/drools-fusion.html>

² <http://esper.codehaus.org>

³ <http://semanticweb.org/wiki/Tripreso>

This paper presents a more practical approach: Linked Data are imported from external sources by being transformed into events conforming to EVO-Core, a lightweight but generic ontology. CEP engines process such events with the support of a Java API generated for manipulating the EVO-Core ontology. The results of event processing will be RDF-ified again and published on the Web of Data. Under this framework, the integration of ontological reasoning and CEP is achieved without any modifications to the RDF and SPARQL standards.

To demonstrate the workflow of the proposed framework, the development of a simple analytical system for user logs in iServe is used as a running example in this paper. iServe [12] is a platform for publishing Semantic Web Service (SWS) descriptions as Linked Data. It is notable that all the data in iServe, including logs of the creation and removal of services, are pure RDF. Therefore, the underlying repository of iServe can be regarded as an external data source having its own schema. In addition, Drools Fusion is used as the example event processing engine, due to its ability to deal with both event streams and clouds.

The rest of this paper is organized as follows: Section 2 reviews the recent work on Semantic CEP. Section 3 summarizes the workflow of the proposed framework at both design time and runtime. Section 4 and Section 5 respectively discuss two critical issues regarding the framework: the event modelling and generation. Section 6 sketches the architecture of the implemented prototype. Section 7 demonstrates the use of the framework. Finally, Section 8 concludes the paper and introduces our future research objectives.

2 Related Work

Earlier research on semantic event modelling is presented in [1], which proposes an approach to reveal the semantics of events by means of classification, aggregation, generalization and association. As a result, a knowledge representation scheme for events is developed to describe complex events, especially the relationships between them. However, the paper only presents theoretical work, and does not touch upon the processing of semantic events.

SQL-like and algebra-based event languages are designed to specify the semantics of events [4, 6]. Nevertheless, they also lack solid support from event processing engines. With advances in Semantic Web technologies, more practical solutions to Semantic CEP are proposed [14, 3]. In [14], the authors present a formal extension to the RDF data model, which is called Time-Annotated RDF (TA-RDF). The main idea is to attach a timestamp to each group of RDF triples. The authors of [3] extend the standard SPARQL query language by adding four binary temporal operators: `SEQ`, `EQUALS`, `OPTIONAL-SEQ`, and `EQUALS-OPTIONAL`, so that Semantic CEP can be done by executing Event Processing SPARQL (EP-SPARQL) queries. Obviously, both these two solutions require modifications to and optimizations of SPARQL query engines and RDF repositories. In contrast, the framework proposed in this paper is based on standard semantic modelling and query language, i.e. RDFS and SPARQL, as well as a mature and well-used CEP engine.

3 Workflow

At design time, the work to build up an event processor consuming Linked Data includes three stages listed below. Additionally, it also involves some trivial tasks such as configuring the connectors to RDF repositories, setting the options of the CEP engine, etc.

- **Event (Stream) Modelling:** Define domain specific events and split them into different streams. Write SPARQL `CONSTRUCT` queries, which are executed at runtime to produce event streams.
- **Code Generation:** Automatically generate Java API for manipulating events and the event processing results, using the code generator provided by RDFReactor. It may also require auxiliary coding work to be done manually, e.g. translating instances of Java Calendar into time in the format of milliseconds.
- **Rule Definition:** Define rules for event processing. If needed, develop some helping functions of, for instance, accessing external SPARQL endpoints on the Web of Data, publishing rule-based reasoning results as Linked Data, etc.

In brief, the results of work done at design time contain: i) an application oriented event model, ii) Java libraries for manipulating event model and processing results, iii) the specification of rules for event processing. All of these are inputs to the runtime modules, which work following the flow illustrated by Fig. 1. Event streams are formed by executing SPARQL `CONSTRUCT` queries



Fig. 1. The runtime workflow diagram.

against certain sets of Linked Data on a regular basis, and, when necessary, SPARQL `DESCRIBE` queries may also be executed to make a snapshot of the concerned entities. If a RDF triple store like BigOWLIm⁴ offers a notification mechanism, the data transformation will be performed when being notified by the triple store. With the runtime supports of RDFReactor⁵ and RDF2Go⁶, the generated event streams are sent to Drools Fusion in the form of Java objects. Drools Fusion performs rule-based reasoning, as well as temporal reasoning over the received Java objects, then RDF-ifies the results and saves into the assigned semantic repository by invoking the Java APIs generated at design time.

⁴ <http://www.ontotext.com/owlim/big>

⁵ <http://semanticweb.org/wiki/RDFReactor>

⁶ <http://semanticweb.org/wiki/RDF2Go>

4 Event Model

The proposed framework aims at bringing together Linked Data and rule-based CEP engines, so the following requirements and issues are taken into account when building the conceptual model of events.

- **Usability:** This is made up of two aspects: i) following the Linked Data principles, especially the ability to be interlinked to RDF triples on the Web of Data; ii) ease of being fed into CEP engines such as Drools Fusion.
- **Extendibility:** The event model should be in the form of a generic ontology, rather than a domain specific one, and must be easy to apply to different application areas.
- **Expressiveness:** The model may be able to describe complex events and event streams, as well as the timing, causality, aggregation and hierarchy of events.
- **Simplicity:** Some applications powered by CEP engines, e.g. Business Activity Monitoring (BAM), are real-time or quasi real-time systems. Thus, light-weight semantics of the event model should minimize the impact of ontological reasoning on performance.

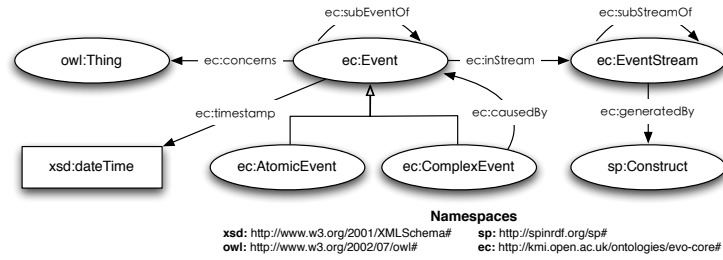


Fig. 2. EVO-Core: a generic event ontology.

As visualized by Figure 2, Event Ontology Core (EVO-Core) is defined in RDF Schema to fulfill the presented demands. It contains four key concepts:

- **Event**, is a concept on the highest level of abstraction and the common ancestor of **AtomicEvent** and **ComplexEvent**. A particular property **timestamp** is used to specify the time when the event happens, and **subEventOf** is for modelling the hierarchy of events. The values of property **concerns** are external links to instances of **owl:Thing**.
- **AtomicEvent**, refers to an instantaneous occurrence of interest.
- **ComplexEvent**, may be built up from a set of other events that hold certain temporal relationships or satisfy constraint conditions on their attributes. The property **causedBy** captures causality among events.

- **EventStream**, is a timely sequence of individual events that come from a data source. The property `inStream` associates events to streams that come into being by repeatedly executing SPARQL `CONSTRUCT` queries. Here, the queries are expressed using SPARQL Inferencing Notation (SPIN⁷), and stored as instances of `sp:Construct` associated with event streams via `generatedBy`. SPIN is essentially a set of vocabularies for writing SPARQL queries in RDF. This way, machines can carry out further reasoning over queries, such as checking their correctness. Similar to `subEventOf`, the property `subStreamOf` models the hierarchy of event streams.

Efforts have already been made to build the conceptual model of events, and relevant ontologies are found in [13, 11]. However, they are neither general purpose, nor suitable for being processed by CEP engines. The Event ontology originates from research in the digital music area, where an event “is regarded as an arbitrary classification of a space/time region, by a cognitive agent” [13].

From an artificial intelligent perspective, it is believed that an event may have five key features: a location, a time, active agents, factors and products. Thus, the Event ontology defines one property for each of the five features. However, at least in some cases like the iServe logging analysis, location might not be applicable. Because the range of the time property is arbitrary temporal entities, i.e. either Instant or Interval defined in the OWL Time Ontology [10], events cannot be sent directly to CEP engines like Drools Fusion, which only can handle timestamps in milliseconds. As for the other three properties, i.e. agent, factor and product, if necessary, they can be defined as sub-properties of concerns in EVO-Core. In general, everything that induces or relates to the occurrence of an event will be a value of concerns, even the instances of Instant and Interval mentioned above. Another weakness of Event ontology is the lack of a facility for expressing complex events and event streams. It only provides one property, called `sub_event`, to capture the hierarchy of events, and nothing for causal relationships among events.

Event Ontology (EVO) is another representative event model [11]. It is the outcome of our previous work, but also the cornerstone of EVO-Core. The difference between these two ontologies is that the target application area of the EVO ontology is Semantic Business Process Management (SBMP), especially Business Process Analysis (BPA). EVO just extends the Core Ontology for Business Process Analysis (COBRA) with several concepts related to states and transitions of process or activity instances, i.e. seven Process Monitoring Events and twelve Activity Monitoring Events, so as to track the running status of business activities. In short, EVO-Core is a generalized version of EVO, further enhanced by the ability to describe complex events and event streams.

5 Event Generation

As highlighted earlier, Linked Data are published by independent providers, following different schemas that are not designed for being processed by CEP

⁷ <http://spinrdf.org/spin.html>

engines. Therefore, it requires ontological mappings [8] between the schema of Linked Data and the event model. This section explains the process of mapping and translating Linked Data into events through an example of iServe logging system [2]. Listing 1 shows part of the schema definition of iServe user logs.

```
log:LogEntry a rdfs:Class ; log:hasDateTime time:Instant ;
              log:hasAction log:Action ; log:hasAgent foaf:Agent .
time:Instant a rdfs:Class .
time:inXSDDateTime rdfs:domain time:Instant ; rdfs:range xsd:dateTime .
log:Action a rdfs:Class .
log:ServiceRepositoryLogEntry a log:LogEntry .
log:ItemCreation rdfs:subClassOf log:Action ; log:createdItem log:Item .
log:ItemDeleting rdfs:subClassOf log:Action ; log:deletedItem log:Item .
```

Listing 1. RDF schema of iServe log entries.

Extensions are made to the EVO-Core ontology to describe the iServe user behaviours. Two new concepts **ServiceCreated** and **ServiceDeleted** are defined as sub-classes of **AtomicEvent**. Moreover, two sub-properties of concerns, **concernsAgent** and **concernsService** are also added to the ontology. The **ServiceCreated** event happens when a new service is uploaded to iServe, whereas **ServiceDeleted** occurs when it is removed by an iServe user. As the name implies, **concernsAgent** and **concernsService** respectively keep the user's FOAF ID and the URI of the service. Finally, **LoggingSystemError** is defined as a **ComplexEvent**, which can be caused by not only a wrong temporal relationship, i.e. a **ServiceDeleted** event occurred before a **ServiceCreated** event, but also by the absence of the corresponding **ServiceCreated** event of a **ServiceDeleted** event.

Formulae (1) and (2) formalize the morphism from the RDF schema of iServe logs to the extended EVO-Core ontology:

$$\begin{aligned} & \text{ServiceRepositoryLogEntry}(l) \wedge \text{hasAction}(l, a) \wedge \text{hasAgent}(l, g) \wedge \text{hasDateTime}(l, i) \wedge \\ & \text{inXSDDateTime}(i, t) \wedge \text{ItemCreation}(a) \wedge \text{createdItem}(a, s) \supseteq \text{ServiceCreated}(e) \wedge \\ & \text{concernsAgent}(e, a) \wedge \text{concernsService}(e, s) \wedge \text{timestamp}(e, t) \end{aligned} \quad (1)$$

$$\begin{aligned} & \text{ServiceRepositoryLogEntry}(l) \wedge \text{hasAction}(l, a) \wedge \text{hasAgent}(l, g) \wedge \text{hasDateTime}(l, i) \wedge \\ & \text{inXSDDateTime}(i, t) \wedge \text{ItemDeleting}(a) \wedge \text{deletedItem}(a, s) \supseteq \text{ServiceDeleted}(e) \wedge \\ & \text{concernsAgent}(e, a) \wedge \text{concernsService}(e, s) \wedge \text{timestamp}(e, t) \end{aligned} \quad (2)$$

Listing 2 elaborates the SPARQL query written according to formula (1). It is not hard to come up with a similar one from the other mapping formula, which is left out from this paper due to space limitations.

```
CONSTRUCT {
  _:v rdf:type ec:ServiceCreated ; ec:timestamp ?time ;
      ec:concernsAgent ?agent ; ec:concernsService ?service ;
      ec:inStream ec:iServeStream .
} WHERE {
  ?entry rdf:type log:ServiceRepositoryLogEntry ; ?entry log:hasAgent ?agent ;
          log:hasAction ?action .
  ?action rdf:type log:ItemCreation ; ?action log:createdItem ?service .
  ?entry log:hasDateTime ?instant .
  ?instant time:inXSDDateTime ?time .}
```

Listing 2. An example of SPARQL query for event generation.

To enable reasoning on the SPARQL queries for ontology translation, the SPARQL query above is converted into the syntax of SPIN (shown in Listing 3), before being stored in an RDF repository. There have been the tool⁸ and online bi-directional converter⁹ between SPARQL and SPIN.

```
_:b1 sp:varName "action"^^xsd:string .
_:b2 sp:varName "service"^^xsd:string .
_:b3 sp:varName "instant"^^xsd:string .
_:b4 sp:varName "agent"^^xsd:string .
_:b5 sp:varName "time"^^xsd:string .
_:b7 sp:varName "entry"^^xsd:string .
[] a sp:Construct ;
sp:templates (
  [sp:object ec:ServiceCreated ; sp:predicate rdf:type ; sp:subject _:b6]
  [sp:object _:b5 ; sp:predicate ec:timestamp ; sp:subject _:b6]
  [sp:object _:b4 ; sp:predicate ec:concernsAgent ; sp:subject _:b6]
  [sp:object _:b2 ; sp:predicate ec:concernsService ; sp:subject _:b6]
  [sp:object ec:iServeStream ; sp:predicate ec:inStream ; sp:subject _:b6]) ;
sp:where (
  [sp:object log:ServiceRepositoryLogEntry ;
   sp:predicate rdf:type ; sp:subject _:b7]
  [sp:object _:b4 ; sp:predicate log:hasAgent ; sp:subject _:b7]
  [sp:object _:b1 ; sp:predicate log:hasAction ; sp:subject _:b7]
  [sp:object log:ItemCreation ; sp:predicate rdf:type ; sp:subject _:b1]
  [sp:object _:b2 ; sp:predicate log:createdItem ; sp:subject _:b1]
  [sp:object _:b3 ; sp:predicate log:hasDateTime ; sp:subject _:b7]
  [sp:object _:b5 ; sp:predicate time:inXSDDateTime ; sp:subject _:b3]) .
```

Listing 3. SPARQL query in SPIN syntax.

Listing 4 outlines a log entry in iServe, against which executing the SPARQL query shown in Listing 2, we can get the first event in Listing 5.

```
log:logEntry1271364976707 a log:ServiceRepositoryLogEntry ;
                           log:hasAction log:action1271364976707 ;
                           log:hasAgent <http://revyu.com/people/dong> ;
                           log:hasDateTime time:instant1271364976707 .

log:action1271364976707 a log:ItemCreation ;
                       log:createdItem service:VEHICLE_PRICE_SERVICE .

time:instant1271364976707 a time:Instant ;
                          time:inXSDDateTime "2010-05-15T20:56:16.707Z"^^xsd:
                          dateTime.
```

Listing 4. A log entry in iServe.

The other two events in Listing 5 are also generated from the iServe system logs, and serve as the examples of `ServiceDeleted` and `LoggingSystemError`.

```
:event101307 a ec:ServiceCreated ;
              ec:timestamp "2010-05-15T20:56:16.707Z"^^xsd:dateTime ;
              ec:concernsAgent <http://revyu.com/people/dong> ;
              ec:concernsService service:VEHICLE_PRICE_SERVICE ;
              ec:inStream ec:iServeStream .

:event107470 a ec:ServiceDeleted ;
              ec:timestamp "2010-04-15T20:56:38.377Z"^^xsd:dateTime ;
              ec:concernsAgent <http://revyu.com/people/dong> ;
              ec:concernsService service:VEHICLE_PRICE_SERVICE ;
              ec:inStream ec:iServeStream .
```

⁸ <http://www.topquadrant.com/products/SPIN.html>

⁹ <http://sparqlpedia.org/spinrdfconverter.html>


```

:event108946 a ec:LoggingSystemError ;
             ec:timestamp "2010-04-15T20:56:41.546Z"^^xsd:dateTime ;
             ec:causedBy event101307 ; ec:causedBy event107470 ;
             ec:inStream ec:iServeStream .

```

Listing 5. Some events in the iServe logging system.

With the helps of RDFReactor on code generation and the Drools' capability of manipulating Java objects, what we need to do to empower Drools Fusion to deal with the **ServiceCreated** and **ServiceDeleted** events is just adding the following declarations to the DRL file:

```

declare ServiceCreated
    @role(event)
    @timestamp(timestampInMills)
end

declare ServiceDeleted
    @role(event)
    @timestamp(timestampInMills)
end

```

Here, **@role** tells the CEP engine the type of the declaring entity, and **@timestamp** tells which attribute will be used as the source of occurrence time of events.

6 Implementation

Fig. 3 depicts the overall architecture of the prototype developed for proof of concept. BigOWLIm serves as the repository for the RDF triples of events. RDF2Go provides a unified interface to various triple (and quad) stores, through which RDFReactor goes to get the access to the repository. The event processor runs on top of the generated Java API for both EVO-Core ontology and analysis results. It consists of three components, namely, event generator, Drools Fusion and timer. Their functionalities have been described in Section 5. Finally, the Linked Data provider, which is also implemented based on the generated Java API, offers several interfaces for clients, including HTML, Linked Data and a SPARQL endpoint. End users can browse the event processing results with a plain HTML browser or with an RDF browser, both supported seamlessly by the server through content negotiation. Third-party applications can interact with the prototype through the SPARQL endpoint.

7 Case Study

This section concentrates on a case study on applying the proposed framework to the analysis of iServe logs. First, we are going to answer the question: who are the top ten active users of iServe so far? Here, active users refer to those who own the most services in iServe. To this end, two rules (shown in Listing 6) are defined respectively for processing **ServiceCreated** and **ServiceDeleted** events. Upon submission of a new service, the event generator will put an instance of **ServiceCreated** into the **iServe Stream**. As the reaction to this event, the event processor finds the user identified by the value of **concernsAgent**, increases

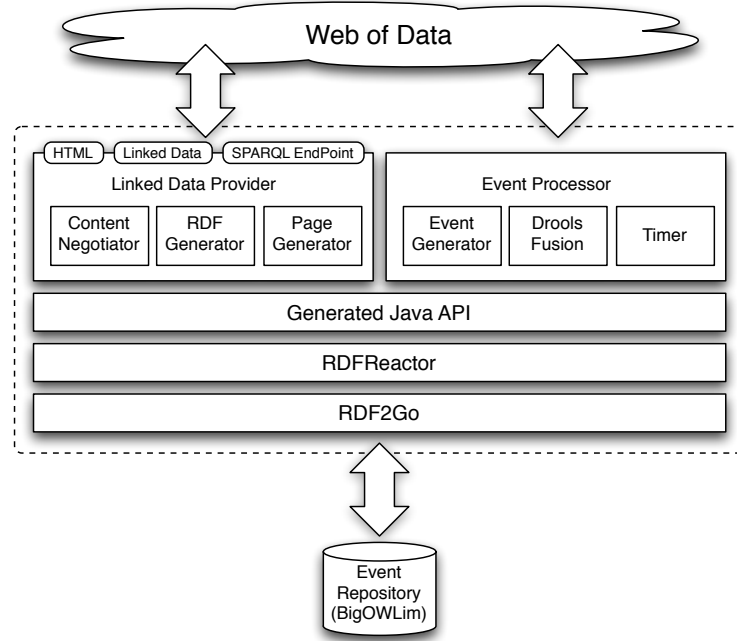


Fig. 3. The overall architecture of prototype.

the number of services he/she uploaded by one, and updates the time of his/her last action on iServe. Correspondingly, when **ServiceDeleted** event happens, the number of uploaded service will decrease by one, and the last action time will update in the same way.

```

rule "Service Created in iServe"
when $e:ServiceCreated( ) from entry-point "iServe Stream"
then String agent=$e.getAllConcernsAgent().next().toString();
  LepHelper.get().increaseUploadedServiceNumber(agent);
  LepHelper.get().updateLastActionTime(agent,
    $e.getAllTimestamp().next());
end
rule "Service Deleted in iServe"
when $e : ServiceDeleted( ) from entry-point "iServe Stream"
then String agent = $e.getAllConcernsAgent().next().toString();
  LepHelper.get().decreaseUploadedServiceNumber(agent);
  LepHelper.get().updateLastActionTime(agent,$e.getAllTimestamp().next());
end
rule "Logging System Error Detecting"
when $e : ServiceDeleted( ) from entry-point "iServe Stream"
and ( $e1 : ServiceCreated( this after $e &&
  concernsService == $e.concernsService ) from entry-point "iServe stream"
or not( $e1 : ServiceCreated( concernsService == $e.concernsService )
  from entry-point "iServe Stream" ))
then
  LepHelper.get().createLoggingSystemErrorEvent($e, $e1);
end
  
```

Listing 6. Event processing rules definition.

Secondly, in order to guarantee the accuracy of analysis results and to detect errors in logging system, we define the third rule of Listing 6. It detects the complex event `LoggingSystemError`, which occurs when a `ServiceDeleted` has a missing `ServiceCreated`, or when a `ServiceDeleted` event happens before the `ServiceCreated` of the same service. Note that, for the ease of understanding, Drools rule attributes, e.g. `no-loop`, `salience`, `lock-on-active`, etc., are omitted from Listing 6. In addition, `LepHelper` is a Java class wrapping various methods for accessing SPARQL endpoints, invoking EVO-Core API, and persisting the analysis results as RDF triples.

The schema below is a simple vocabulary for describing the analysis results:

```
:Agent rdf:type rdfs:Class .
:uploadedServiceNumber a rdf:Property ; rdfs:domain :Agent ;
                        rdfs:range xsd:nonNegativeInteger .
:lastActionTime a rdf:Property ; rdfs:domain :Agent ;
                rdfs:range xsd:dateTime .
```

According to the schema, it is not hard to come up with a SPARQL `SELECT` query retrieving ten users ordered by the number of services they have uploaded to `iServe`:

```
SELECT ?agent ?number ?time WHERE {
  ?agent a ia:Agent ; ia:uploadedServiceNumber ?number ;
        ia:lastActionTime ?time .
} ORDER BY DESC(?number) DESC(?time) LIMIT 10
```

Especially, when the numbers are the same, they will be ordered by the last time they accessed `iServe`. The query results are displayed in Fig. 4. For privacy reason, some of the FOAF IDs have been concealed.

Agent	Number	Time
<http://iserve.kmi.open.ac.uk>	"1694"^^xsd:int	2010-07-01T15:18:08.000Z
<http://revyu.com/people/dong>	"85"^^xsd:int	2010-06-30T20:14:38.703Z
<http://[redacted].myopenid.com/foaf.rdf>	"28"^^xsd:int	2010-08-16T13:08:39.883Z
<http://[redacted].com>	"23"^^xsd:int	2010-05-02T11:40:19.454Z
<http://www.[redacted]>	"14"^^xsd:int	2010-04-27T13:14:32.477Z
<http://[redacted].myopenid.com/>	"8"^^xsd:int	2010-03-26T17:17:40.667Z
<http://kmi.open.ac.uk/people/member/[redacted]>	"7"^^xsd:int	2010-03-22T15:48:08.958Z
<http://[redacted].myopenid.com/>	"7"^^xsd:int	2010-03-19T14:50:24.854Z
<http://people.kmi.open.ac.uk/rdf.php>	"4"^^xsd:int	2010-06-21T16:45:40.330Z
<http://foafbuild[redacted].com/>	"1"^^xsd:int	2010-05-20T13:22:46.340Z

Fig. 4. The screenshot of query result.

8 Conclusions and Future Work

In this paper, we present a practical way in which Linked Data can be fed into CEP engines. EVO-Core, a lightweight but generic ontology, is built to describe

events in RDF, and SPARQL-based ontological mapping technique is adopted to transform Linked Data into events conforming to EVO-Core. We also introduce the workflow of developing an application equipped with the event processor. The development of a simple analytical system of iServe logs shows that the proposed framework is feasible.

Our future work will focus on the publication of analysis results according to SDMX-RDF [7]. We will also try to run a public registry for Linked Data sources, which can be used as the origin of events, together with the corresponding SPARQL queries for ontological translation.

Acknowledgements This work was funded by the EU project SOA4All (FP7-215219).

References

1. Adi, A., Botzer, D., Etzion, O.: Semantic Event Model and Its Implication on Situation Detection. In: Proceedings of the 2000 European Conference on Information Systems (ECIS) (2000)
2. Álvaro Rey, G., Cerizza, D., Di Matteo, G., Ripa, G., Turati, A., Villa, M.: SOA4All Project Deliverable: D.2.7.1 - Recommender System First Prototype (2009)
3. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: Stream Reasoning and Complex Event Processing in ETALIS. Semantic Web – Interoperability, Usability, Applicability (Under Review), <http://www.semantic-web-journal.net/sites/default/files/swj90.pdf>
4. Brenna, L., Demers, A.J., Gehrke, J., Hong, M., Ossher, J., Panda, B., Riedewald, M., Thatte, M., White, W.M.: Cayuga: a high-performance event processing engine. In: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data. pp. 1100–1102. ACM (2007)
5. Etzion, O.: Semantic Approach to Event Processing. In: Proceedings of the 2007 inaugural international conference on Distributed event-based systems (DEBS). pp. 139–139. ACM, New York, NY, USA (2007)
6. Gyllstrom, D., Wu, E., Chae, H.J., Diao, Y., Stahlberg, P., Anderson, G.: SASE: Complex Event Processing over Streams. In: Proceeding of the 3rd Biennial Conference on Innovative Data Systems Research (CIDR). pp. 407–411 (2007)
7. Hausenblas, M., Halb, W., Raimond, Y., Feigenbaum, L., Ayers, D.: SCOVO: Using Statistics on the Web of Data. In: Proceedings of the 6th European Semantic Web Conference (ESWC). Lecture Notes in Computer Science, vol. 5554, pp. 708–722. Springer (2009)
8. Kalfoglou, Y., Schorlemmer, M.: Ontology mapping: the state of the art. The Knowledge Engineering Review 18(1), 1–31 (2003)
9. Luckham, D.C.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2001)
10. Pan, F., Hobbs, J.R.: Temporal Aggregates in OWL-Time. In: Russell, I., Markov, Z. (eds.) Proceedings of the 18th International Florida Artificial Intelligence Research Society Conference (FLAIRS). pp. 560–565. AAAI Press (2005)

11. Pedrinaci, C., Domingue, J., de Medeiros, A.K.A.: A Core Ontology for Business Process Analysis. In: Hauswirth, M., Koubarakis, M., Bechhofer, S. (eds.) Proceedings of the 5th European Semantic Web Conference (ESWC). Lecture Notes in Computer Science, Springer Verlag, Berlin, Heidelberg (June 2008)
12. Pedrinaci, C., Liu, D., Maleshkova, M., Lambert, D., Kopecky, J., Domingue, J.: iServe: a Linked Services Publishing Platform. In: Proceedings of Ontology Repositories and Editors for the Semantic Web at the 7th Extended Semantic Web Conference (ESWC) (2010)
13. Raimond, Y., Abdallah, S.: The Event Ontology (2007), <http://purl.org/NET/c4dm/event.owl>
14. Rodriguez, A., McGrath, R., Liu, Y., Myers, J.: Semantic Management of Streaming Data. In: Proceedings of the 2nd International Workshop on Semantic Sensor Networks (SSN) at the 8th International Semantic Web Conference (ISWC). pp. 80–95 (Oct 2009)
15. Teymourian, K., Paschke, A.: Enabling Knowledge-based Complex Event Processing. In: Proceedings of the 2010 EDBT/ICDT Workshops. pp. 1–7. ACM, New York, NY, USA (Mar 2010)